Task1:

Determine if String Halves Are Alike

Aim:- To determine the two halves of a given even-length string are alike by checking if both halves contain an equal number of vowels.

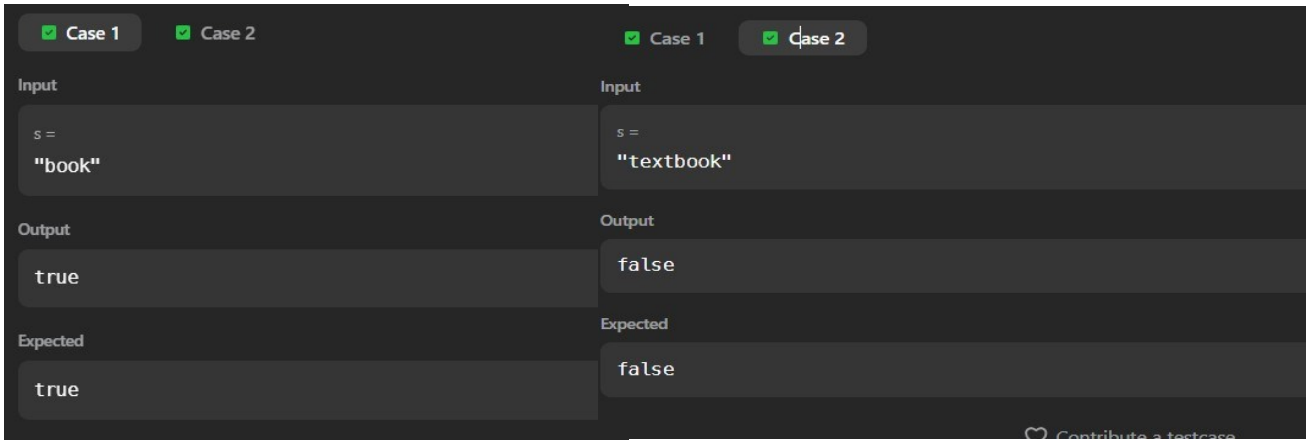
Algorithm:-

1. Start
2. Read the input string s**.**
3. Find the midpoint of the string using mid = length / 2.
4. Initialize two counters countA and countB to zero.
5. Define a string containing all vowels (both uppercase and lowercase).
6. Traverse from index **0** to mid - 1:
7. If the character in the first half is a vowel, increment countA.
8. If the corresponding character in the second half is a vowel, increment countB.
9. Compare countA and countB.
10. If both counts are equal, return true; otherwise, return false.
11. stop

Program:-

```
class Solution {
public boolean halvesAreAlike(String s) {
      int n = s.length(), count = 0;
      String vowels = "aeiouAEIOU";
      for (int i = 0; i < n / 2; i++)
         if (vowels.indexOf(s.charAt(i)) >= 0) count++;
      for (int i = n / 2; i < n; i++)
 if (vowels.indexOf(s.charAt(i)) >= 0) count--;        return
count == 0;
   }
}
```

Output:-



Result:-Thus, To determine the two halves of a given even-length string are alike by checking if both halves contain an equal number of vowels is are successfully executed.

Task2:

LAPIN

Aim:- To check whether a given string is a **lapindrome**. A string is said to be a lapindrome if, after splitting it into two halves (ignoring the middle character if the length is odd), both halves contain the same characters with the same frequency.

Algorithm:-

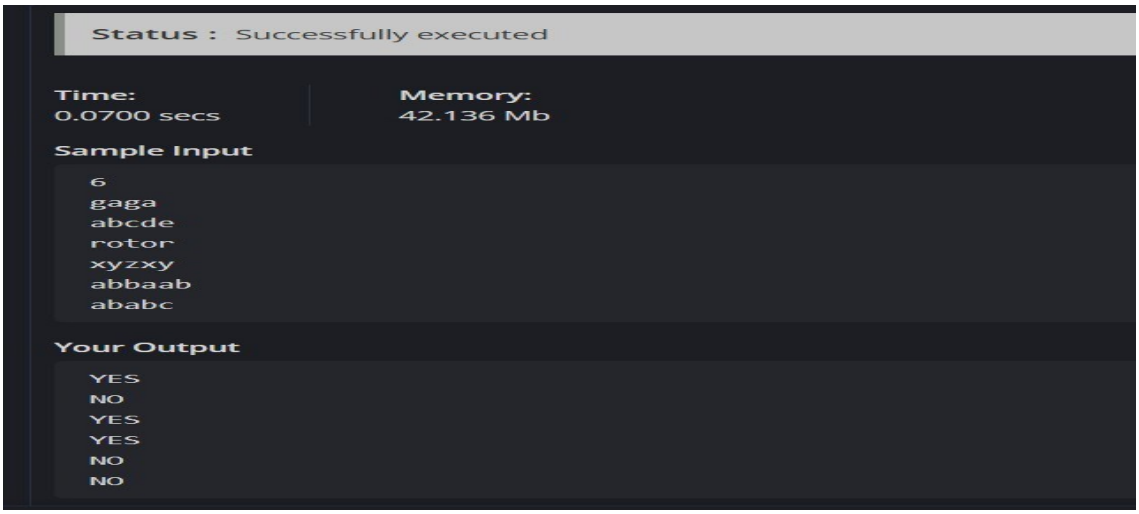
1. Start.
2. Read the number of test cases T.
3. For each test case:
    - Read the string S. o Find the length n of the string. o Calculate the middle index mid = n / 2.
4. Create two frequency arrays of size 26 (for lowercase English letters).
5. Count the frequency of characters in the first half of the string.
6. Count the frequency of characters in the second half of the string:
    - If the string length is odd, skip the middle character.
7. Compare both frequency arrays:
    - If all frequencies match, the string is a lapindrome.
    - Otherwise, it is not a lapindrome.
8. Print "YES" if the string is a lapindrome, else print "NO".
9. Stop

Program :-

```
import java.util.*;


class Codechef {

   public static void main(String[] args) throws java.lang.Exception {

      Scanner sc = new Scanner(System.in);

      int T = sc.nextInt();

      while (T-- > 0) {

         String s = sc.next();
```

```
        int n = s.length();
        int[] freq1 = new int[26], freq2 = new int[26];
        int mid = n / 2;
        for (int i = 0; i < mid; i++) freq1[s.charAt(i) - 'a']++;
        for (int i = (n % 2 == 0 ? mid : mid + 1); i < n; i++) freq2[s.charAt(i) - 'a']++;
        boolean isLapindrome = true;
        for (int i = 0; i < 26; i++) {
            if (freq1[i] != freq2[i]) {
                isLapindrome = false;
                break;
            }
        }
        System.out.println(isLapindrome ? "YES" : "NO");
    }
    sc.close();
  }
}
```

Output:-



Result:-Thus,The program successfully identifies whether each given string is a lapindrome or not by comparing the character frequencies of its two halves and displays “YES” for lapindrome strings and “NO” for non-lapindrome strings is are successfully executed.

Task3:

Compare the Triplets

Aim:- To implement a java code for Compare the Triplets using loops.

Algorithm:-

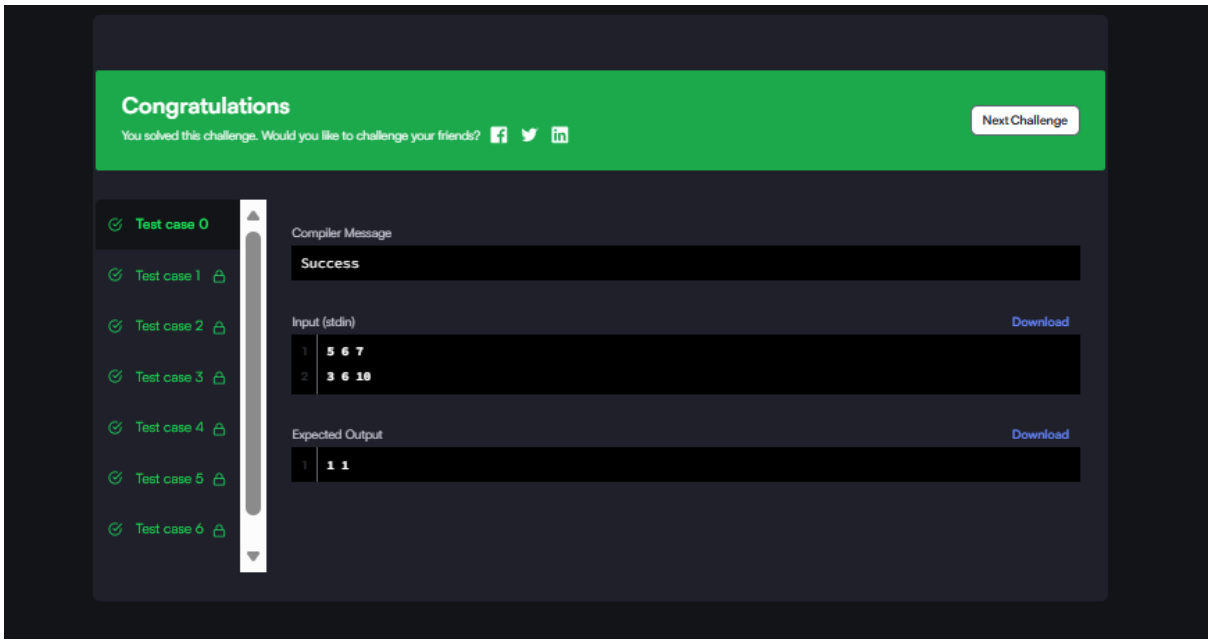
1. **Input:**
    - Read Alice's triplet a[0], a[1], a[2].
    - Read Bob's triplet b[0], b[1], b[2].
2. **Initialize scores:**
    - alice = 0
    - bob = 0
3. **Comparison loop:** For each index i from 0 to 2:
    - If a[i] > b[i], increment alice.
    - Else if a[i] < b[i], increment bob.
    - Else (equal), do nothing.
4. Print Alice's score and Bob's score as [alice, bob].

Program:-

```
import java.util.*;
public class Solution {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int[] a = new int[3];
      int[] b = new int[3];
      for (int i = 0; i < 3; i++) a[i] = sc.nextInt();
      for (int i = 0; i < 3; i++) b[i] = sc.nextInt();
      int alice = 0, bob = 0;
      for (int i = 0; i < 3; i++) {
         if (a[i] > b[i]) alice++;
         else if (a[i] < b[i]) bob++;
      }
```

```
        System.out.println(alice + " " + bob);
}}
```

Output:-



Result:- Thus, To implement a java code for Compare the Triplets using loops is are successfully executed.

Task4:

Contains Duplicate

Aim:- To determine whether an integer array contains any duplicate values. Return true if at least one value appears more than once, otherwise return false.

Algorithm:-

1. Input: An integer array nums.
2. Initialize a data structure:
3. Use a HashSet to store unique elements.
4. Iterate through the array:
5. For each element n in nums:
    a. Try to add n to the set.
    b. If adding fails (element already exists), return true.
6. If loop completes:
7. No duplicates found → return false.

Program:-

```
import java.util.*;
import java.util.stream.*;
class Solution {
   public boolean containsDuplicate(int[] nums) {
      return Arrays.stream(nums).distinct().count() < nums.length;
   }
}
```

Output:-

Result:-Thus, To determine whether an integer array contains any duplicate values. Return true if at least one value appears more than once, otherwise return false verified successfully.

Task5:

Time Conversion

Aim:- To convert a given time in 12-hour AM/PM format into 24-hour military time format.

Algorithm:-

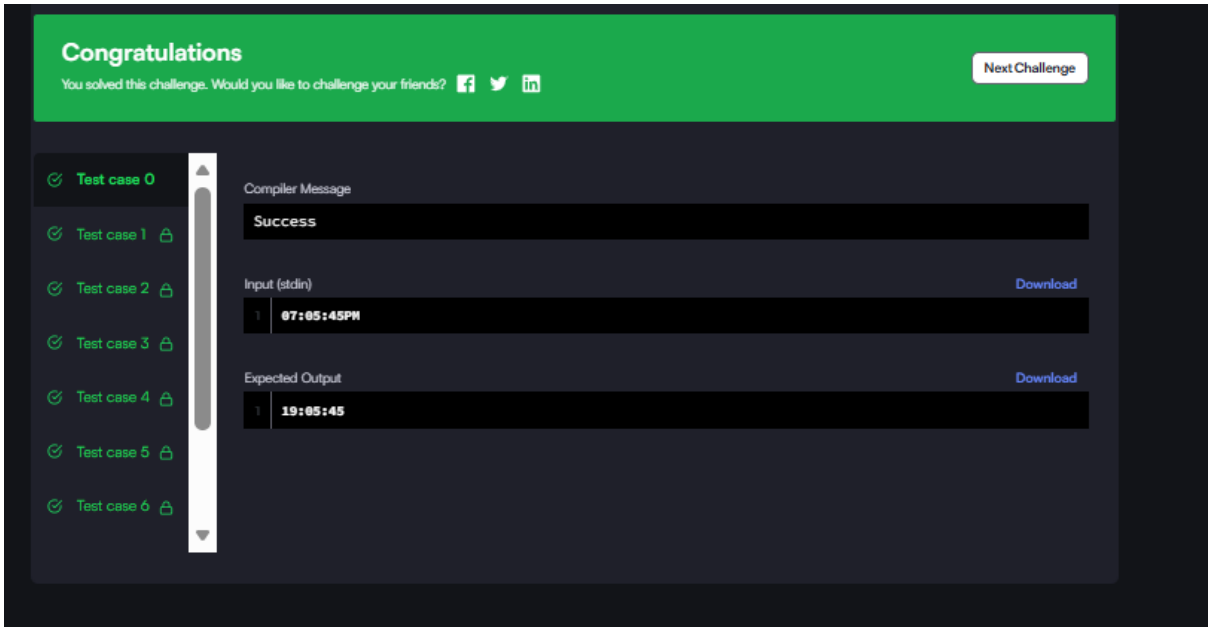
1. **Input:** A string s representing time in the format hh:mm:ssAM or hh:mm:ssPM.
2. **Extract components:**
    - Hours (hh), minutes (mm), seconds (ss), and the period (AM/PM).
3. **Conversion rules:**
    - If the period is AM:
        - If hh == 12, set hh = 00.
        - Else keep hh unchanged.
    - If the period is PM:
        - If hh != 12, add 12 to hh.
4. **Format result:**
    - Return the string in HH:MM:SS format (24-hour).

Program:-

```
import java.util.*;
class Solution {
   public static String timeConversion(String s) {
      String period = s.substring(8);
      int hour = Integer.parseInt(s.substring(0, 2));
      if (period.equals("AM")) {
         if (hour == 12) hour = 0;
      } else {
         if (hour != 12) hour += 12;
      }
      return String.format("%02d:%s", hour, s.substring(3, 8));
   }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println(timeConversion(s));
    }
}
```

Output:-



Result:-Thus, To convert a given time in 12-hour AM/PM format into 24-hour military time format is are successfully executed.

Task6:

Move Zeroes

Aim:- To move all zeroes in an integer array to the end while keeping the relative order of the non-zero elements intact. The operation must be done in-place (no extra copy of the array).

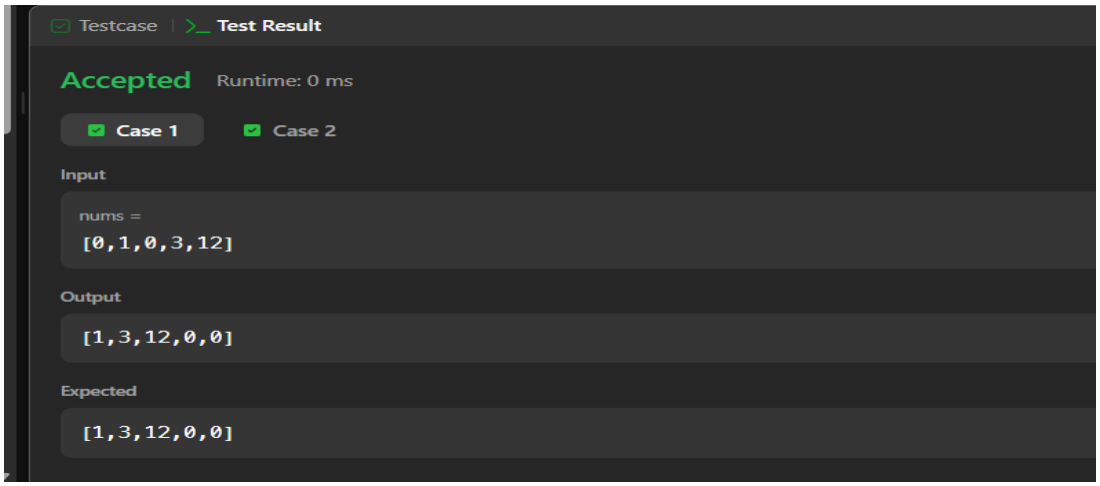
Algorithm:-

1. Input: An integer array nums.
2. Initialize a pointer: pos = 0 (tracks the position to place the next non-zero).
3. First pass:
4. Iterate through the array.
5. For each non-zero element, assign it to nums[pos] and increment pos.
6. Second pass:
7. From index pos to the end of the array, fill with zeroes.
8. The modified array with all non-zero elements in order followed by zeroes.

Program:-

```
class Solution {
    public void moveZeroes(int[] nums) {
        int pos = 0;
        for (int n : nums) if (n != 0) nums[pos++] = n;
        while (pos < nums.length) nums[pos++] = 0;
    }
}
```

Output:-



Result:-Thus, To move all zeroes in an integer array to the end while keeping the relative order of the non-zero elements intact. The operation must be done in-place (no extra copy of the array) is are successfully executed.

Task7:

Diagonal Difference

Aim: -To calculate the absolute difference between the sums of the primary diagonal and the secondary diagonal of a square matrix.

Algorithm:-

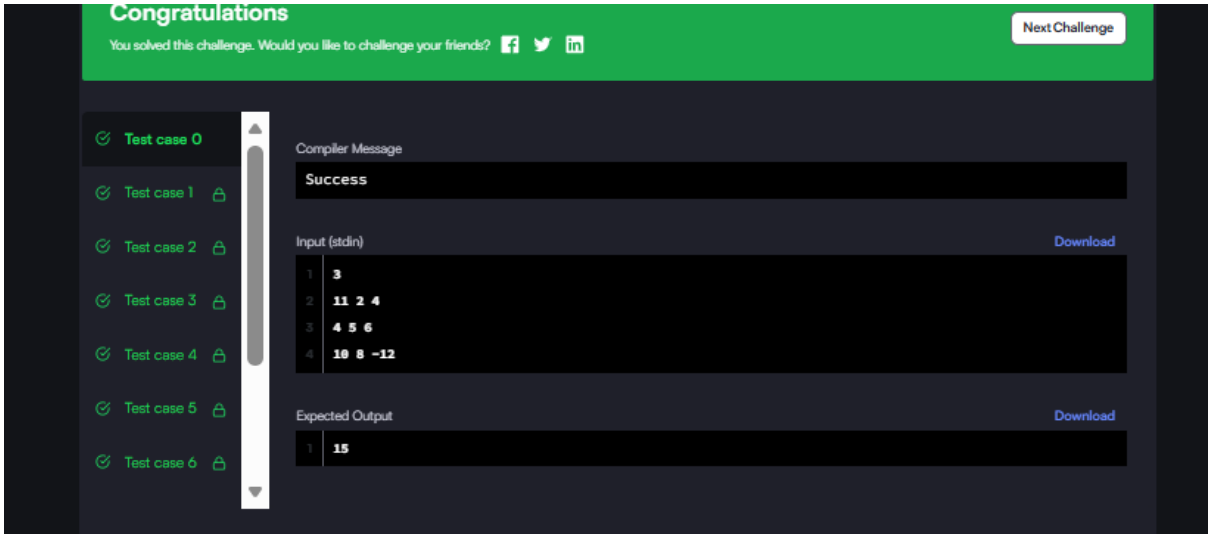
1. **Input:**
    - An integer n (size of the matrix).
    - An n × n matrix arr.
2. **Initialize sums:**
    - primary = 0
    - secondary = 0
3. **Loop through rows:** For each index i from 0 to n-1:
    - Add arr[i][i] to primary (primary diagonal).
    - Add arr[i][n-1-i] to secondary (secondary diagonal).
4. **Compute difference:**
    - result = ”primary – secondary”
5. Print or return result.

Program:-

```
import java.util.*;


class Solution {
   public static int diagonalDifference(int[][] arr) {
      int n = arr.length, primary = 0, secondary = 0;
      for (int i = 0; i < n; i++) {
         primary += arr[i][i];
         secondary += arr[i][n - 1 - i];
      }
      return Math.abs(primary - secondary);
   }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[][] arr = new int[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                arr[i][j] = sc.nextInt();
        System.out.println(diagonalDifference(arr));
    }
}
```

Output:-



Result:-Thus, To calculate the absolute difference between the sums of the primary diagonal and the secondary diagonal of a square matrix is are successfully executed.

Task8:

Transpose Matrix

Aim:-To return the transpose of a given 2D integer matrix. The transpose is formed by flipping the matrix over its main diagonal, switching rows with columns.

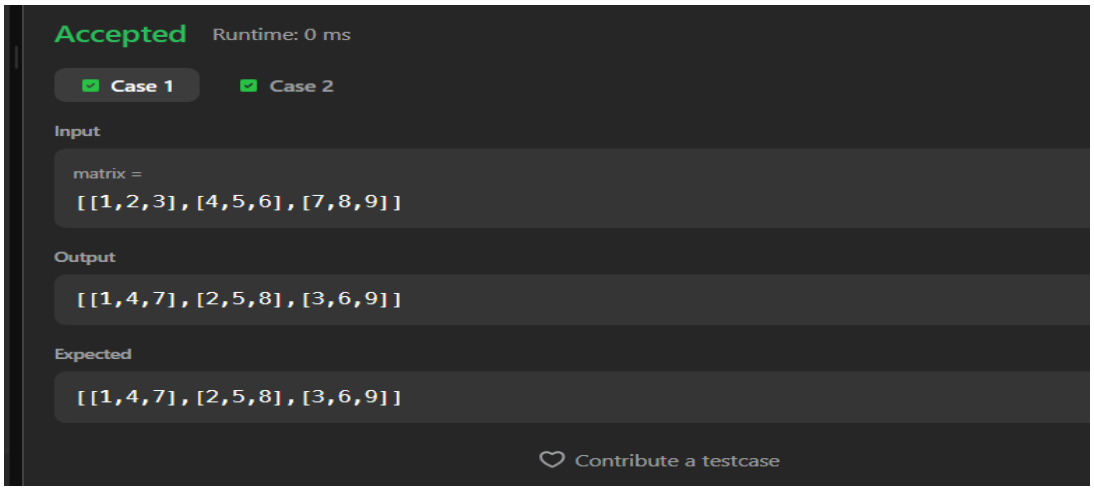
Algorithm:

1. **Input:** A 2D integer array matrix of size m × n.
2. **Initialize:** A new 2D array result of size n × m.
3. Loop through elements:
4. For each element at position (i, j) in the original matrix, place it at (j, i) in the result.
5. Return the result matrix.

Program:-

```
class Solution {
    public int[][] transpose(int[][] matrix) {
        int m = matrix.length, n = matrix[0].length;
        int[][] result = new int[n][m];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                result[j][i] = matrix[i][j];
        return result;
    }
}
```

Output:-



Result: Thus, To return the transpose of a given 2D integer matrix. The transpose is formed by flipping the matrix over its main diagonal, switching rows with columns is are successfully executed.